支援教師分身的智慧型程式語言教學平台

董少桓, 林宗德, 沈維倫, 錢傳明, 沈勇嘉
雲林科技大學資訊管理系
{tungsh, g9323803, g9223740, u9123009, u9323316}@yuntech.edu.tw

摘要

對於教授程式語言的課程而言，所學的程式語言與開發環境不適合初學者學習、學生與老師的互動少、缺乏個別化的指導，這些都是教學上常見的問題。「支援教師分身的智慧型程式語言教學平台」便是為了解決這些問題而產生的。此教學平台支援適合初學者學習的程式語言—Scheme 的教學課程，讓學生以主動式的「learning by doing」方式學習。搭配「循序漸進式練習題組」之教學策略，此平台可以半自動提示學生程式的語法和邏輯錯誤。而且能透過網路提供老師給予學生個別輔導建議的資訊，讓老師能夠快速的幫助學生解決問題。就學生而言，就像有一位專屬的「教師分身」在個別指導一樣。

關鍵詞： 數位學習、程式語言教學、Scheme、互動式教學平台

一、簡介

(一) 研究背景與動機

長久以來，大學內程式設計課程所教授的程式語言，常常有跟著流行走的現象。從 Cobol、Fortran、C、C++，一直到近期流行的 Java，都可以觀察到這種歷史的趨勢。至於當前流行的某個程式語言是不是一定適合初次學習程式設計的學生學習，則少有人過問 [5]。

以 C、C＋＋或 Java 而言，這些程式語言都是用來解決工商業界實際的複雜問題而設計的。相較於 dynamic type 支援 read-eval-print loop 開發方法的程式語言，C、C＋＋與 Java 都屬於 static type 而且需要編譯之後才能執行的「互動程度低」的程式語言。同時又為了支援大型軟體計畫的開發，這些程式語言的 IDE 也包括了許多初學者不會用到的功能。使用這種「專業級」、「互動程度低」的程式語言與開發環境來教授初次學習程式設計的學生，勢必會產生許多諸如語法複雜、錯誤信息難以理解、學習進度緩慢等妨礙學習的問題 [5, 6]。其實對初學者而言，能透過快速互動，迅速的學習問題解決與邏輯思考能力，要比學習複雜的語法與開發工具來的重要的多。因此，選擇一個互動程度高而且又能夠以簡單的語法，來訓練初學者邏輯思考及程式設計能力的程式語言，對提升程式語言的教學效益來說，便非常的重要。

除了程式語言與開發環境的不適當，傳統缺乏與學生互動的授課方式，也降低了學生的學習效率。原因是學習程式設計最有效的方式，不是在上課時聽講解，而是在聽了講解後能透過練習，實際的體會講授的內容。這種主動式「learning by doing」的學習方式，要比被動的聽課，更適合程式語言的學習。不過，只是給學生練習的機會還是不夠 — 初學者還需要「個別化的指導」以有效的挪開許多阻礙學生學習的問題。以初學者在電腦教室上課的情境而言，一位老師需要同時指導數十位甚至六、七十位同學，這種師生比過低的現象，使得許多同學所面對的程式設計問題，不能得到及時的指示或引導，降低了學習效率，還可能導致學習興趣的喪失。

為了解決這些程式語言的教學問題，過去數十年，已經有相當多的 Intellgent Tutoring System (ITS) 被開發出來 [2, 8]。基本上這些 ITS 系統可以概略的分為單機版與網路版兩類。學生與單機版系統的互動方式，完全是透過安裝在學生電腦上的軟體提供的。網路版的 ITS 系統則容許學生在任何時間、任何地點與伺服器端的系統或其他同時上線的學習同伴進行互動式學習。

然而在這兩種系統中間，還有一種尚未被嘗試的方式。這種方式是提供給在同一時間、一起利用電腦教室上課的老師與學生使用的。這種系統扮演的角色，不單是幫助學生，更是幫助老師能更有效的成為所有上課學生的個別指導者。也就是這個系統的目的是希望能透過網路提供老師充分的學生學習狀況與個別輔導建議的資訊，好讓老師能夠快速的幫助學生解決撰寫程式的過程中所碰到的問題。就學生而言，這個系統能夠讓每一位上課的同學都覺得好像有一位專屬的「教師分身」在個別的指導學生的學習一樣。與傳統一個老師講課數十位學生聽課的被動式學習方式相比，這個教學系統能夠讓每一位同學在「learning by doing」的主動式學習過程中，得到更多個別化的指導。

(二) 研究目的

「支援教師分身的智慧型程式語言教學平台」便是在這樣的背景的下所提出的研究。我們將

開發一個 Scheme 程式語言教學系統。Scheme 是一個能夠與學習者高度互動的 dynamic type 的程式語言，它的特色是可以用少數幾個簡單的語法、明確的語意與高度互動的特性，快速的訓練學生的邏輯思考能力 [4]。這些基本的語法與語意，雖然基本但並非低能。這些基本的語法與語意，還同時具備了足以支援功能式、命令式與物件導向式程式設計觀念的高度「表達能力」。這些特色加上其高互動的特性，使得學生能迅速的得到學習的回饋與成就感，增進其學習的效率。

「支援教師分身的智慧型程式語言教學平台」之研究目標是為 Scheme 開發具備學習能力的半自動偵錯及提供提示功能的協助教學系統。這個系統能夠提供解答學生程式設計問題的建議選項。在系統無法產生適當建議的時候，教師可以用人工的方式，將解決問題的建議或提示，輸入系統並且透過網路快速地傳遞給學生。這樣的安排不但可以讓教師在有限的上課時間中，增加與學生互動的能力；而且能不斷的累積系統的知識庫，增加其偵錯及提供提示的能力，朝著成為全自動系統的目標邁進。

完成「支援教師分身的智慧型程式語言教學平台」對改進程式語言的教學而言，可以有以下許多的好處：

1. 學生可以透過經過設計的程式設計練習，一次的練習一個能夠日漸累積學習效益及複雜度的程式，並且經由與系統及老師的互動，提升學習成效。
2. 教師可以透過監控學生的答題狀況，掌握學生的學習進度及困難。
3. 改變傳統被動式一對多的授課方式，成為每一位學生都能分享一位「教師分身」的主動式 learning by doing 的教學方式。
4. 提供一個師生能夠共同參與的平台，使得程式語言的教學內容與知識能夠透過練習題庫及問題收集不斷的累積與豐富。
5. 透過課程內容的累積，可以將國外一流大學教授 Scheme 語言與程式設計的相關知識內容封裝起來，以便導入到國內其他大專院校甚至高中職的相關課程，有助於提昇國內程式設計課程的教學品質。

除了適用於 Scheme 程式語言的學習，我們相信這種提供教師分身功能的網路教學系統，也適合許多能夠以「learning by doing」的方式學習並且需要利用電腦才能完成作業的科目。例如：C、C++ 以及 Java 之學習、UML 的設計、SQL 的練習等。因此，在本研究的研究過程中，我們將以設計軟體元件與 framework 的方式，讓其他科目也能夠使用我們開發出來的教學平台。

二、相關研究

無獨有偶的，在美國加州柏克萊大學的電機與電腦系的 UCB-WISE 計畫，亦有與本研究類似的課程規劃及相關的教學軟體設計 [3]。從其教學網站及所發表的論文中我們發現本研究與 UCB-WISE 計畫有以下相似的地方：

1. 同樣都是使用 Scheme 語言當作基本程式設計課程的程式語言。
2. 都在轉換傳統一對多的 lecture 型態的授課方式為全面使用電腦教室的 learning by doing 的授課方式。
3. 同樣都計畫能將成果推廣至其他大學、社區學院或高中。

然而 UCB-WISE 計畫與本研究不同的地方在於：

1. UCB-WISE 較著重於支援同學彼此間的互動功能。例如使用特殊設計的討論區以支援所謂「腦力激盪」的功能。
2. 教師的主要功能是訓練其他真人擔任教學助理、幫助特別需要幫助的同學以及更正學生的錯誤觀念。
3. 沒有提供教師分身的功能。

整體而言 UCB-WISE 的設計是以學生的需求為主，而本研究的重點是以老師的需求為主。造成這種設計方向差異的原因或許是 UCB 基本程式設計課程的師生比高達 1 比 5。在公布於 UCB-WISE 網站上的資料中發現，有一個學期的 Scheme 課程有三個 lab sections，這三個 sections 的學生人數分別是 22、18 及 10，而且每個 section 都有一位老師及 2-3 位的教學助理。在這種高師生比的環境，自然沒有「教師分身」的需要。然而在國內，我們相信能夠有效的幫助老師，是學生能得到充分照顧的先決條件，因為老師的工作就是教學生。

在美國麻省理工學院的電機與資訊科學系中也有一門以 Scheme 當作入門語言與主要程式設計工具的課程。這門叫做「Structure and Interpretation of Computer Programs」的課程在麻省理工學院的「開放式課程網頁」及其中文翻譯版的網站中也有列入(http://www.twocw.net) [1]。然而，這門課網站中的內容只包含了課程大綱、教學投影片、線上教科書、考試題目、作業解答等靜態的內容，而沒有能夠支援學生與系統及教師互動的輔助教學功能。

另外一個與本研究關連度很高的相關研究是 Lisp Tutor 及類似的 Intelligent Tutoring Systems (ITS) [2, 8]。這類系統可以概分為單機版的系統與網路版的系統。早期單機版的系統可以說是一個專門負責教授 Lisp 遞迴程式設計的專家系統。而網路版的系統除了支援單機版的功能外，還容許學生透過網路互相成為學習同伴，或是使用主機上的專

家系統進行學習。這類系統的特色是能夠達成完全不需要教師監控便能幫助學生學習的「全自動」輔助學習的目標。然而，「全自動」的最大困難是需要花費大量的人工才能建置。因此只能提供數量不多的練習，而不能完整的支援整個課程的需要。為了改進這個缺點，本研究所要建置的教學系統，將提供教師一個簡易的輸入練習題目及解答的界面，以供系統自動產生具備基本偵錯及諮詢能力的資料庫。接著師生可以透過實際的使用，逐步增加系統的知識庫。因此，在系統初步上線時，我們期待教師需要以人工回答學生問題的頻率會比較高。然而在經過不斷的使用後，教師便可以更多的利用系統自動產生的諮詢或提示來回答學生的問題。這種「自動學習」的功能是「支援教師分身的智慧型程式語言教學平台」與傳統 Lisp-based ITS 的主要的不同之處。

最後一個與本研究相當有關連的研究是一個用來學習 C、C++與 Java 的基本語法與語意的程式語言學習網站。這個由 Turingscraft 公司所經營的網站提供了教授程式語言的教師與學生一個教學管理與學習的平台。教師可以透過該網站察看學生的答題狀況，而學生也可以利用該網站所提供的題目，練習撰寫符合 C、C++或 Java 語法與語意的程式片斷。這個網站的特色是提供約二百個答案簡短又可以自動偵測答案是否正確的程式設計練習。這些練習可以用來訓練學生對程式語言基本語法與語意的熟練程度。由於答案簡短，所以學生可以快速的與該網站互動並得到學習回饋，再加上教師可以將學生的答題狀況算入學生的成績，使得該網站能夠以商業化的方式進入許多美國大學的校園。然而這個網站還沒有做到的地方是訓練學生學習諸如 recursion、higher-order functions 等比較需要深入思考的程式設計觀念。雖然如此，該網站還是驗證了「支援教師分身的智慧型程式語言教學平台」的實用價值及可行性。

三、系統設計

本研究所建置的「支援教師分身的智慧型程式語言教學平台」的特色是它能夠透過師生的使用而不斷的進步。就智慧型偵錯與指導學生如何撰寫程式的能力而言，可以因著紀錄學生的問題與教師的回覆，而不斷的提升。因此，建置一個輸入、儲存與展示練習題目的環境，然後透過實際的使用以不斷的改進系統的功能與使用的便利性，便是完成研究目標的主要方法。

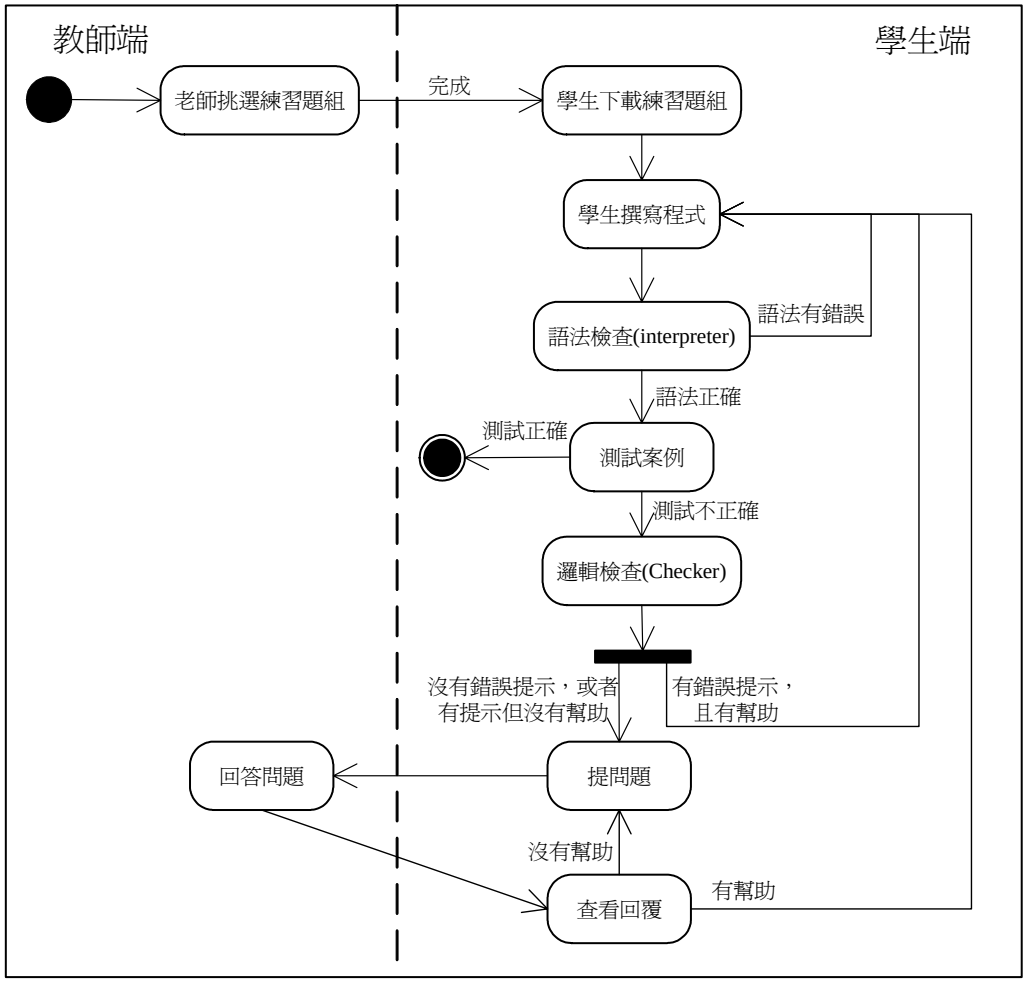

圖 1: 系統活動圖

以功能面來區分，「支援教師分身的智慧型程式語言教學平台」之系統架構可分為教師端與學生端兩個主要的部份。圖 1 為系統的活動圖，用來表示老師端與學生端的運作流程。

1. 教師端

教師端的功能有二：編輯或挑選練習題組，以及回答問題。在老師可以挑選練習題組之前，老師必須事先編輯好練習題組之內容。老師編輯練習題組時需要輸入練習題之題目描述、解答、和測試案例等資料。題目描述是一組文字敘述該練習題之題目；解答是該練習題的正確解答程式；而測試案例則是數組用來測試學生解答是否正確的案例程式。以下是一個簡單的範例：

```
題目描述：
    試寫出一支 remove 程式可以將指定的字元從串列中移除。
解答：
    (define remove
      (lambda (a ls)
        (cond
          ((null? ls) '())
          ((eq? a (car ls)) (remove a (cdr ls)))
          (else (cons (car ls) (remove a (cdr ls)))))))
測試案例：
    (remove 'a '(a b a c d a)) => (b c d)
    (remove ‘a ‘()) => ()
```

當老師完成練習題組的編輯之後，可以挑選練習題組，提供給學生開始作答。若學生在作答過程中需要老師的協助，老師可以透過網路回答學生之問題。在老師回答學生問題時，可以選擇將訊息儲存起來，以便下次遇到類似問題時可以快速地回覆。在系統剛開始上線時，老師可能以人工輸入訊息的方式回答問題的機會較多。但是隨著時間的累積，資料庫內收集到的問題和回覆訊息也愈來愈多，老師以人工輸入的方式也會愈來愈少。老師只要從資料庫裡面挑選適當的訊息即可快速地回答問題。

2. 學生端

老師在挑選練習題組之後，系統發佈題目給學生下載並開始做答。當學生完成作答的程式後，可以利用 Scheme 的 interpreter 先檢查程式的語法是否正確。若語法上有錯誤，則由 interpreter 回覆錯誤訊息給學生，回去重新修改程式。若語法上沒有錯誤，則進入邏輯錯誤之診斷。邏輯錯誤診斷之方式為將事先存在於資料庫中的測試案例丟入學生之程式中執行。若執行結果與預期結果相同，則表示邏輯上應該無誤，該學生之作答即可結束。另一方面，若測試案例之結果有誤，則表示有邏輯上之錯誤。系統會將學生程式送給一個邏輯檢查程式(Checker)進行診斷。若 Checker 有找出邏輯錯誤之原因，其提示訊息將回應給學生。學生可依 Checker 提示之訊息重新回去修改程式。若 Checker 提示之訊息對學生沒有幫助或者是 Checker 找不到錯誤的原因，則學生可以再透過網路提出自己所遭遇到的困難，尋求老師的協助。學生可以選擇依照老師回覆的訊息回去修改程式，或者繼續提問題。

在以上的幾個步驟中，如何讓 Checker 為一個有邏輯錯誤的程式提供有教育意義的提示，無疑是最困難的部分。我們的構想是將正確解答建構出一個分析其邏輯架構的文件。以稍早所提到的 remove 程式當作範例，其解答經過處理之後，應會產生如圖 2 之樹狀結構。

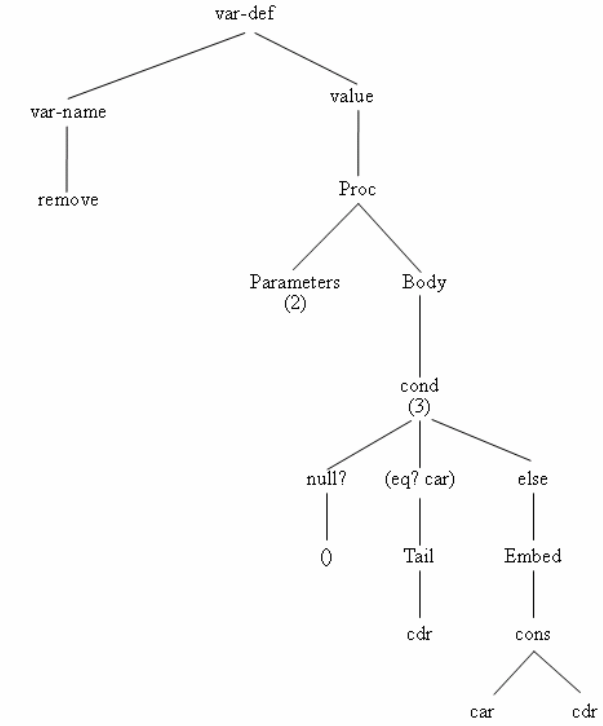

圖 2：程式經過處理後之樹狀結構範例

這棵樹提供了可行解的參數的個數、程式碼中各個條件判斷式所使用到的 procedure 的名稱以及特性。其中 Embed 及 Tail 分別代表 embedded recursion 及 tail recursion 的呼叫。這些資訊可以用來與學生的程式比對並將學生程式中缺少的、多出的或誤用的程序呼叫比對出來，然後產生一些提示傳送給學生。以下面這個有錯誤的程式為例：

```
(define remove
  (lambda (a ls)
    (cond
      ((null? ls) '())
      ((eq? a (car ls)) (remove a (cdr ls)))
      (else (cons (car ls) (remove a ls)))))))
```

Checker 對這個程式的提示是：“Embedded recursive call incorrect: (cons (car ls) (remove a ls))”

四、教學策略

對於程式語言來說，如果完全沒有限制的話，相同的題目可以有很多種的解答來得到相同的預期結果。為了可以將學生的程式儘量控制在有限的範圍內，我們以「循序漸進式程式練習題組」的教學策略來搭配「支援教師分身的智慧型程式語言教學平台」，讓邏輯錯誤判斷與提供有教育意義的提示變得可行。

「循序漸進式練習題組」之教學策略是以一連串的練習題讓學生在「learning by doing」中學習，而這一連串的練習題聯貫起來可以完成一個教學目標。「循序漸進式練習題組」一開始是以一支簡單的程式為基礎，每往下一題只要加入少量的程式碼就可以得到正確解答。當學生完成整個練習題組之後，即可以達到該課程進度之目標。

以下是一個學習 abstraction 概念的練習題組，以一個計算元素個數的程式為範例。這個範例是在課程進度已經進入到大約第 5 週，而且學生已經對於 Scheme 語言和遞迴有基本的認識時所使用。題組中的第 1 支程式 count-a 是在計算一個 list 當中有多少個“a”字元。其解答如下：

```
(define count-a
  (lambda (ls)
    (cond
      ((null? ls) 0)
      (((eq? (car ls) 'a)) (+ 1 (count-a (cdr ls))))
      (else (count-a (cdr ls))))))
```

接下來，題組中的第 2 題 count-a-or-b 是將第一題做一點變化，計算一個 list 當中有多少個字元是屬於“a”或“b”。這個解答只要修改第 1 題中的第 5 行程式碼的前半段即可，如下所示：

```
(define count-a-or-b
  (lambda (ls)
    (cond
      ((null? ls) 0)
      (((or (eq? (car ls) 'a) (eq? (car ls) 'b)) (+ 1
(count-a-or-b (cdr ls))))
      (else (count-a-or-b (cdr ls))))))
```

第 3 題 count-symbol 不只是計算指定的字元，而是可以計算所有字元的個數。其解答與第 2 題之差別也只是在第 5 行的程式碼需要修改而已。如下所示：

```
(define count-symbol
  (lambda (ls)
    (cond
      ((null? ls) 0)
      (((symbol? (car ls)) (+ 1 (count-symbol (cdr
ls))))
      (else (count-symbol (cdr ls))))))
```

第 4 題 count-all 則是比第 3 題的能力更強，可以不用事先指定要計算那一種字元，等到要呼叫此 procedure 時才指定即可。所以 count-all 在經過 abstraction 之後，其適用範圍就更為廣泛了。然而第 4 題的解答也只需要從第 3 題的解答當中增加 1 個參數以及修改第 5 行程式碼即可。如下所示：

```
(define count-all
  (lambda (f ls)
    (cond
      ((null? ls) 0)
      ((f (car ls)) (+ 1 (count-all f (cdr ls))))
      (else (count-all f (cdr ls))))))
```

呼叫 count-all 的方式則是將第 1 個參數帶入一個 function 即可。例如計算一個 list 當中有多少個數字，可以這樣使用：

```
(count-all number? '(a 2 a c 4))
```

這種「循序漸進式練習題組」之教學策略搭配「支援教師分身的智慧型程式語言教學平台」有以下三個特色：第一，老師以引導式的方式教學，深入淺出，學生可以較容易地完成整個題組。以計算元素個數的範例而言，在學生完成整個練習題組之後，即可清楚地了解 abstraction 的意義及方法。第二，將學生的程式限制在某個可能的範圍內。由於每往下一題，學生都只需要依照上一題的解答修改少量的程式碼即可得到正確解答，故學生程式可以限制在某個可能範圍內。也讓邏輯錯誤的判斷與提供有教育意義的提示變得可行。第三，將「循序漸進式練習題組」之教學策略搭配本研究設計之教學平台，學生可以在學習的過程當中隨時得到 Checker 和老師的提示訊息支援。當學生能力提升時，即可減少提示訊息的支援，符合鷹架式教學的理論。

五、實作

「支援教師分身的智慧型程式語言教學平台」之系統架構如同第三章所描述分為教師端與學生端兩個主要部份。教師端的挑選與編輯練習題組功能，本研究採用 Web-based 方式實作，老師只要透過網路即可使用瀏覽器編輯練習題組，如圖 3 所示。當老師編輯完成時，系統將所編輯之練習題資料儲存到資料庫當中，以供學生練習。

學生端的部份，依照本研究之系統設計應包括下列元件：程式編輯器、Scheme 的 interpreter 與執行環境、診斷學生程式邏輯錯誤的 Checker、以及提問題之訊息溝通介面。這些元件的實作可以選擇在伺服器端執行或是在客戶端執行。在客戶端執行的好處是當同時上線的學生過多時，可以減少伺服器的負荷；但缺點是必須將元件下載到客戶端才能執行。在伺服器端執行的好處是學生只要進入網站即可使用，不必另外下載程式；但缺點是伺服器的負荷較重。考量此兩種方式各有其優缺點，因此本研究分別實作出在伺服器端執行與在客戶端執行的學生編輯程式介面。

圖 3：老師編輯練習題之執行畫面

在伺服器端執行的學生編輯程式畫面如圖 4 所示。畫面左邊的樹狀結構列出所有課程內容(lesson)的名稱，其課程內容是依照「循序漸進式練習題組」所編排。每一個課程內容即為一個練習題組，一個題組內可能包含多個練習題。畫面的中間部份顯示的是題組中的練習題，一次顯示一題。學生完成一個練習題之後，即可按“下一題”按鈕進入下一個練習題。當學生完成練習題組中的每一個練習題之後，就結束該課程內容，可以進入下一個課程內容。

圖 4：在伺服器端執行的程式編輯畫面

當學生在程式編輯區撰寫程式完畢之後，需要一個 Scheme 的 interpreter 與執行環境來偵測語法錯誤與執行測試案例。Second Interpreter of Scheme Code (SISC)是一個 Java-based 的 Scheme interpreter [7]，它提供 Scheme 語法的檢查與執行環境。當學生在程式編輯區中完成練習題之後即可按“執行”按鈕，系統會將程式編輯區內的程式送到後端的伺服器以 SISC 檢查語法並執行。若出現語法上的錯誤，SISC 會傳回錯誤之訊息顯示在輸出區(圖 4 的下方)。若編譯之結果沒有發現錯誤，則再將測試案例與學生程式一起丟入 SISC 中執行，看輸出是否與預期結果相同。如果測試結果與預期結果相同，表示學生程式在邏輯上應該也沒有問題(完成作答)。但是，如果測試結果與預期結果不同，則可以按“檢查”按鈕，將學生的程式再丟入 Checker 當中診斷其錯誤的原因。

Checker 是由 Scheme 撰寫而成的程式，老師可因練習題之形態不同而選擇不同的 Checker 來診斷學生的程式(參照圖 3 接近下方處)。不同的 Checker 其診斷方式卻大同小異，主要的概念如同第三章所描述。

圖 5：教師端顯示學生問題之畫面

當學生撰寫程式的過程中碰到阻礙而且系統自動提示之訊息無法幫助學生解決問題時，學生可以按“提問”按鈕，透過網路提出問題給老師。學生提問題時系統將學生傳送之訊息、學生程式碼、以及電腦的 IP 位址記錄下來。教師端即可看到學生所提之問題列表(如圖 5 之後方所示)。老師選擇要回答某一個問題之後，按查看按鈕可出現詳細的訊息內容(如圖 5 之前方所示)，並可看到學生之程式碼。老師了解學生所遭遇到的問題後，可以挑選適合的訊息回覆或者以人工的方式回覆，亦或選擇不回答。

另一方面，在客戶端執行的學生編輯程式介面是將所需的 4 個元件包裝成一個 Java Applet 元件，讓學生從網站下載到客戶端執行。學生只需要在第 1 次進入網站啟動此 Java Applet 時自動下載元件，當下一次學生欲使用同一台電腦啟動 Java Applet 時，便自動使用已存在於本機之 Java Applet 元件啟動，除非偵測到有更新版本的元件才需要重新下載。當老師挑選完練習題組並發佈之後，學生即可啟動 Java Applet 元件，並透過 JDBC 到資料庫中一次取得練習題組中的所有練習題，開始撰寫

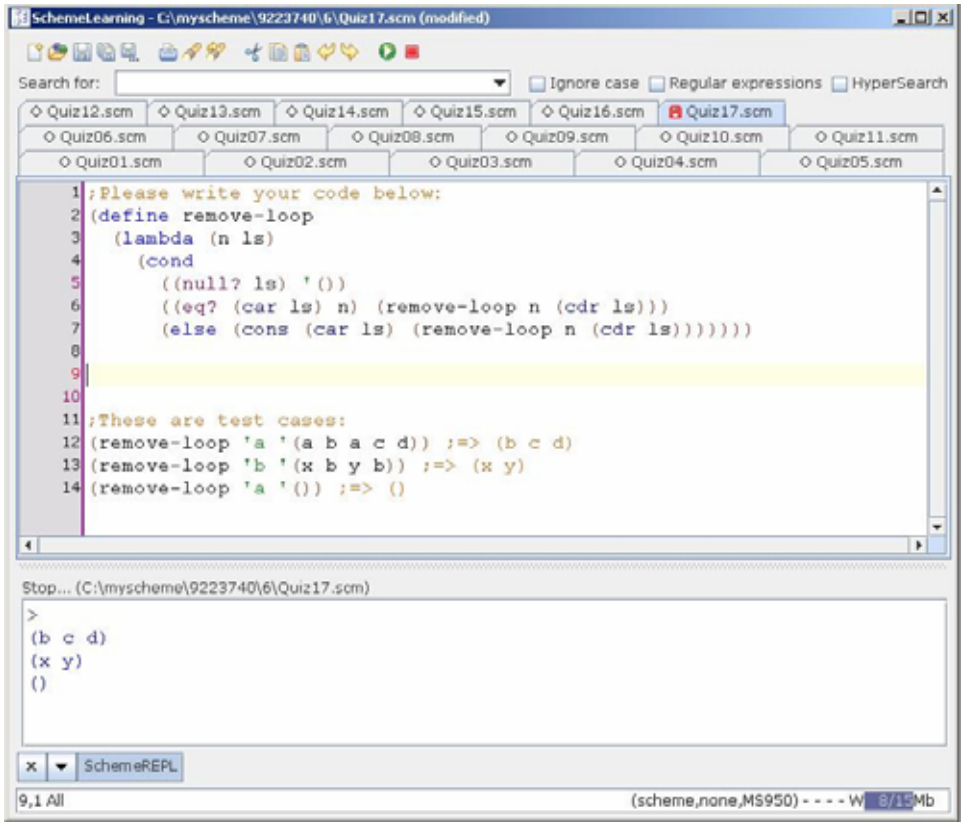

圖 6：在客戶端執行的程式編輯器畫面

程式。

此 Java Applet 元件是以程式編輯器為主體，而將其他部份都以 plug-in 的方式整合到程式編輯器之中。採用 plug-in 方式的原因是為了讓本平台有更大的彈性，將來如果需要發展除了 Scheme 之外的課程(例如：C、C++、Java、UML、SQL…等)，只需要將這些 plug-in 之套件替換掉即可。

jEdit [9] 是一個開放原始碼的程式編輯器，它已經實作了許多程式編輯器所需之功能，包括：「開啟新檔」、「開啟舊檔」、「儲存檔案」、「另存新檔」、「儲存全部檔案」、「剪下文字」、「複製文字」、「貼上文字」、「還原」、「重做」、「搜尋文字」、「程式碼多顏色顯示」、「程式碼自動縮排」、以及「括號配對檢查」等。除此之外，它也提供 plug-in 機制讓開發者能夠自由地擴充其他功能。考量 jEdit 的功能相當符合本研究之所需，故本研究以 jEdit 為基礎來發展 Scheme 語言之程式編輯器。本研究將 jEdit 修改成 Java Applet 版本，並加上 Scheme 語言所使用的「程式碼多顏色顯示」、「程式碼自動縮排」、以及「括號配對檢查」之外掛程式，以方便編寫 Scheme 語言。請參照圖 6 所示。

除了上述的外掛程式之外，程式編輯器也將 SISC、Checker 都當作 plug-in 套件來處理。當學生按下程式編輯器裡面的執行按鈕時，系統便將學生程式丟入 SISC 中檢查語法並執行。其執行步驟與在伺服器端執行之步驟相同，不同的是所有執行的位置都是在客戶端。

為了可以讓老師迅速了解課堂上學生學習的實際狀況，在教師端的介面中提供一個顯示學生作答情形的燈號列表(如圖 7 所示)。綠色的燈號表示該學生已經作答完畢，黃色的燈號表示該學生還沒有完成作答，紅色的燈號號表示該學生是處於閒置的狀態，黑色的燈號表示該學生未開機。

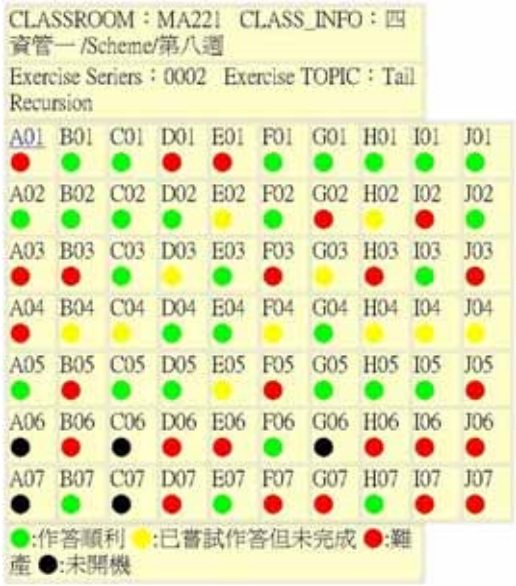

圖 7：學生作答情況燈號列表

燈號的判斷原則是：(1)當學生執行程式完畢，並且通過測試案例的比對無誤，即表示該學生已經作答完畢，應顯示綠燈。(2)當學生已經開始作答，即進入未完成的狀態，應顯示黃燈。(3)當學生已經開始作答，且經過一段時間(時間長短由教師端設定)沒有按“執行”按鈕，即示為閒置狀態，應顯示為紅燈。(4)當學生沒有登入系統時，視為未開機，應顯示為黑燈。

由這個燈號列表老師可以迅速地了解學生作答的情況，以決定是否進入下一個進度的參考。並且可以了解到那一些學生在學習上有障礙，特別是顯示為紅燈的學生，可以給予額外的指導。

六、結論

以教授 Scheme 語言而言，「支援教師分身的智慧型程式語言教學平台」能充分的支援在同一時間、同一地點一起利用電腦教室上課的老師與學生能有效的透過「learning by doing」的學習方式增進學習的效率。在未來我們期待此平台能夠達到不同時間、不同地點也不需要教師監控的「非同步、全自動網路教學」的目標。

以將「支援教師分身的智慧型程式語言教學平台」推廣到其他課程而言。我們期望能夠透過物件化及模組化的原則，更清楚的呈現能夠讓其他相關課程也能夠使用的架構（framework）。並且實際的以 C、C++、Java 或是 UML、SQL 等當作範例，使得更完整的歸納出導入新課程的規範與作法。

誌謝

感謝行政院國家科學委員會對本研究經費的補助。計畫編號為 NSC 94-2213-E- 224-037。

參考文獻

[1] H. Abelson, G.J. Sussman, and J. Sussman, “Structure and interpretation of computer programs”, MIT Press, Boston, MA., 1984

[2] J.R Anderson and E. Skwarecki, “The automated tutoring of introductory computer programming”, Communications of the ACM, Vol. 29, No. 9, pp. 842-849, 1985

[3] M. Clancy, N. Titterton, C. Ryan, J. Slotta and M. Linn, “New roles for students, instructors, and computers in a lab-based introductory programming course”, SIGCSE’03, Reno, Nevada, February, 2003

[4] R.K. Dybvig, “The Scheme Programming Language 2nd Edition”, Prentice Hall PTR, New York, 1996

[5] M. Felleisen, R. B. Findler, M. Flatt and S. Krishnamurthi, “The TeachScheme! project: Computing and programming for every student, Computer Science Education, Vol. 14, No. 1, pp. 55-77, 2004

[6] M. Kolling, B. Quig, A. Patterson and J. Rosenberg, “The BlueJ system and its pedagogy”, Computer Science Education, Vol. 13, No. 4, 2003

[7] S.G. Miller and M. Radestock, SISC for seasoned schemers, http://sisc.sourceforge.net/manual/html/, 2004

[8] W.K. Wong, T.W. Chan, C.Y. Chou, J.S. Heh and S.H. Tung, “Reciprocal tutoring using cognitive tools”, Journal of Computer-Assisted Learning, Vol. 19, No. 4, 2003

[9] (n.d.), jEdit Development, http://www.jedit.org/